NBSI v1.0 External Architecture

Engineering Specification

Build Instructions for the External Version

Complete implementation guide including node lifecycle, structural library, session graph, SEM operator, and APIs

Field	Value
Document type	Engineering specification — build instructions
Audience	Backend / ML engineers implementing NBSI from the white paper
Scope	Full external system: lifecycle engine, structural library, session graph, SEM, APIs
Language	Python 3.11+ (reference implementation)
New in this spec vs NBSI v4.0	Node lifecycle (observation/stabilisation/reactivation), structural library with generalisation constraint, lifecycle-aware session management
Status	Pre-publication draft 2026

0. System Overview

The external version is a software stack in which every component of the NBSI architecture is explicit, inspectable, and independently testable. Nothing is implicit inside a neural network. The structural node library, session graph, SEM operator, and lifecycle engine are separate modules with defined interfaces.

Module	Component	Persistent?	New in this spec?
Lifecycle	ObservationNode, stabilisation threshold, reactivation detector	Structural layer only	YES — core addition
Structural Library	StructuralNodeLibrary: pattern-only nodes, generalisation constraint	YES — across all sessions	YES — replaces InvariantLibrary
Ingestion	Chunker, Multi-LLM Extractor, Agreement Scorer	No	No
Session Graph	ConceptGraph, ConceptNode, ConceptEdge (base/effective split)	No — session only	No
SEM Operator	sem_propagate, sem_confirm, sem_rollback	No — session only	No
Reasoning	Conductivity (geometric mean), Beam Search, Query Anchor	No	No
Ensemble	Stability measurement, dual-graph intersection	No	No
Session	NBSISession: full lifecycle orchestration	No	Extended with lifecycle
API	FastAPI REST layer	No	Extended with lifecycle endpoints

THE CRITICAL NEW COMPONENT

The node lifecycle is the addition that separates this specification from NBSI v4.0. Every other component exists in the v4.0 engineering spec. The lifecycle engine — observation nodes, stabilisation, reactivation — and the structural library with the generalisation constraint are new. Build these after validating that the v4.0 components work correctly.

1. Environment and Project Structure

1.1 Dependencies

Package	Version	Purpose
Python	3.11+	Runtime
networkx	3.2+	Session graph data structure
sentence-transformers	2.6+	Node/edge/speculation embedding
scipy	1.12+	Wasserstein distance for structural diversity
numpy	1.26+	Array operations
openai	1.x	GPT-4o extractor LLM
anthropic	0.25+	Claude extractor LLM
google-generativeai	0.5+	Gemini extractor LLM
fastapi + uvicorn	0.110+	REST API
pydantic	2.x	Config and schema validation
pytest	8.x	Test runner

1.2 Project Layout

project layout

```
nbsi/
  config.py                 # All parameters with defaults
  lifecycle/                # NEW — core addition
    __init__.py
    observation_node.py     # ObservationNode dataclass
    lifecycle_engine.py     # Stabilisation, reactivation logic
    structural_library.py   # StructuralNodeLibrary
  graph/
    node.py                 # ConceptNode
    edge.py                 # ConceptEdge (base_weight / effective_weight)
    concept_graph.py        # ConceptGraph
  ingestion/
    chunker.py              # Text -> overlapping chunks
    extractor.py            # Multi-LLM node/edge extraction
    agreement.py            # Cross-LLM agreement scoring
  sem/
    speculative_node.py     # SpeculativeNode
    propagator.py           # sem_propagate / confirm / rollback
  reasoning/
    conductivity.py         # Geometric mean path scoring
    traversal.py            # Beam search, query anchor
    ensemble.py             # Stability + dual-graph intersection
  session/
    session.py              # NBSISession (extended with lifecycle)
    manager.py              # SessionManager
```

```
  api/
    routes.py               # FastAPI routes
    schemas.py              # Pydantic models
tests/
  test_lifecycle.py         # NEW — observation, stabilisation, reactivation
  test_sem.py
  test_conductivity.py
  test_session.py
  test_ensemble.py
```

2. The Node Lifecycle (New Component)

This section specifies the new components not present in NBSI v4.0. Build and test these first before integrating with the session graph.

2.1 ObservationNode

```python
# nbsi/lifecycle/observation_node.py
from dataclasses import dataclass, field
from typing import Optional
import uuid

@dataclass
class ObservationNode:
    """
    A provisional pattern candidate. Expensive: demands active attention.
    Transitions to StructuralNode when stabilisation_score >=
STABILISATION_THRESHOLD.
    Pruned if it never stabilises within MAX_OBSERVATION_SESSIONS.
    """
    node_id: str = field(default_factory=lambda: str(uuid.uuid4()))
    pattern_embedding: Optional[list] = None  # 768-dim centroid of seen
instances
    confirmation_count: int = 0               # Times pattern was matched
    diversity_score: float = 0.0              # Wasserstein diversity of host
graphs
    stabilisation_score: float = 0.0          # Composite: confirmation x
diversity
    sessions_active: int = 0                  # Sessions in which this node fired
    example_labels: list = field(default_factory=list)  # Last 5 labels seen (not
stored permanently)

    def update_pattern(self, new_embedding: list, label: str):
        """Update centroid embedding with new instance. Do not store the
instance."""
        if self.pattern_embedding is None:
            self.pattern_embedding = new_embedding
        else:
            # Running average — instances are not stored, only the centroid
updates
            n = self.confirmation_count
            self.pattern_embedding = [
                (self.pattern_embedding[i] * n + new_embedding[i]) / (n + 1)
                for i in range(len(self.pattern_embedding))
            ]
        self.confirmation_count += 1
        self.example_labels = (self.example_labels + [label])[-5:]  # Ring
buffer, not accumulation

    def compute_stabilisation_score(self, diversity: float) -> float:
        """
        Stabilisation = confirmation_count (normalised) x structural diversity.
        High confirmation alone is not enough — must appear in structurally
diverse sessions.
```

```
        """
        normalised_conf = min(1.0, self.confirmation_count / 10.0)
        self.diversity_score = diversity
        self.stabilisation_score = normalised_conf * diversity
        return self.stabilisation_score
```

2.2 StructuralNode

python

```python
# nbsi/lifecycle/structural_library.py
from dataclasses import dataclass, field

@dataclass
class StructuralNode:
    """
    A stabilised processing pattern. Near-zero attention cost.
    Fires silently when content matches. Reactivated only by anomaly.
    Contains NO content — only pattern geometry.
    """
    node_id: str
    pattern_embedding: list          # 768-dim centroid. No instances stored.
    firing_confidence: float = 1.0   # Increases with confirmation, decreases
with anomalies
    session_count: int = 0           # Sessions in which this node fired
    diversity_score: float = 0.0     # Structural diversity of sessions it
appeared in
    reactivation_count: int = 0      # How many times anomaly has triggered
reactivation
    is_generalised: bool = True      # False = entity-specific, must not be
promoted

    # Attention mode: 'infrastructure' (normal) or 'active' (during reactivation)
    attention_mode: str = 'infrastructure'

    def reactivate(self):
        self.attention_mode = 'active'
        self.reactivation_count += 1

    def return_to_infrastructure(self):
        self.attention_mode = 'infrastructure'

    def check_generalisation(self, entity_specificity_score: float, threshold:
float = 0.3) -> bool:
        """
        A node is entity-specific if it fires predominantly for one entity's
content.
        entity_specificity_score: fraction of firings from a single entity [0,
1].
        If > threshold, mark as not generalised. Must not promote.
        """
        self.is_generalised = entity_specificity_score < threshold
        return self.is_generalised
```

2.3 The Lifecycle Engine

```python
# nbsi/lifecycle/lifecycle_engine.py
import math
import numpy as np
from scipy.stats import wasserstein_distance
from itertools import combinations
from sklearn.metrics.pairwise import cosine_similarity
from nbsi.lifecycle.observation_node import ObservationNode
from nbsi.lifecycle.structural_library import StructuralNode,
StructuralNodeLibrary
from nbsi.config import Config

class LifecycleEngine:
    def __init__(self, library: 'StructuralNodeLibrary', embedder, config:
Config):
        self.library = library
        self.embedder = embedder
        self.config = config
        self._observation_nodes: dict[str, ObservationNode] = {}

    # ── Stage 1: Observation ──────────────────────────────────────
    def observe(self, label: str, embedding: list) -> str:
        """
        Called when content does not match any structural node well.
        Returns the observation_node_id.
        """
        # Check if an existing observation node is close enough to update
        for obs in self._observation_nodes.values():
            if obs.pattern_embedding is not None:
                sim = float(cosine_similarity([embedding],
[obs.pattern_embedding])[0][0])
                if sim > self.config.OBSERVATION_MERGE_THRESHOLD:  # default 0.88
                    obs.update_pattern(embedding, label)
                    return obs.node_id
        # New observation node
        obs = ObservationNode()
        obs.update_pattern(embedding, label)
        self._observation_nodes[obs.node_id] = obs
        return obs.node_id

    # ── Stage 2: Stabilisation ────────────────────────────────────
    def evaluate_for_stabilisation(self, obs_id: str,
                                   session_degree_distributions:
list[np.ndarray]) -> bool:
        """
        Called after each session. Returns True if node promoted to structural.
        Requires: confirmation_count >= MIN_CONFIRMATIONS AND diversity >
DIVERSITY_THRESHOLD.
        """
        obs = self._observation_nodes.get(obs_id)
        if obs is None:
            return False
        if obs.confirmation_count < self.config.MIN_CONFIRMATIONS:  # default 5
            return False
```

```
        if len(session_degree_distributions) < 2:
            return False
        pairwise = [wasserstein_distance(d1, d2)
                    for d1, d2 in combinations(session_degree_distributions, 2)]
        diversity = float(np.mean(pairwise))
        score = obs.compute_stabilisation_score(diversity)
        if score >= self.config.STABILISATION_THRESHOLD:  # default 0.4
            self._promote(obs)
            return True
        return False

    def _promote(self, obs: ObservationNode):
        """Promote observation node to structural node. Content-free."""
        structural = StructuralNode(
            node_id=obs.node_id,
            pattern_embedding=obs.pattern_embedding,  # Centroid only — no
instances
            session_count=obs.sessions_active,
            diversity_score=obs.diversity_score,
        )
        self.library.add(structural)
        del self._observation_nodes[obs.node_id]

    # ── Stage 3: Reactivation ──────────────────────────────────────────
    def check_for_reactivation(self, content_embedding: list) -> list[str]:
        """
        Called during session processing when content has low match to all
structural nodes.
        Returns list of structural node IDs that should be reactivated.
        A node is reactivated when content is close enough to match (sim >
LOW_THRESHOLD)
        but not close enough to fire confidently (sim < HIGH_THRESHOLD).
        This is the anomaly zone — the eclipse region.
        """
        reactivated = []
        for node in self.library.nodes.values():
            if node.pattern_embedding is None:
                continue
            sim = float(cosine_similarity([content_embedding],
[node.pattern_embedding])[0][0])
            low  = self.config.REACTIVATION_LOW_THRESHOLD   # default 0.45
            high = self.config.REACTIVATION_HIGH_THRESHOLD  # default 0.75
            if low < sim < high:  # Anomaly zone: matches but imprecisely
                node.reactivate()
                reactivated.append(node.node_id)
        return reactivated

    def refine_reactivated(self, node_id: str, content_embedding: list):
        """
        Refine a reactivated structural node's pattern centroid.
        Uses the anomalous content to sharpen the node boundary.
        Content is not stored — only the centroid moves.
        """
        node = self.library.nodes.get(node_id)
        if node is None or node.attention_mode != 'active':
            return
```

```
        # Adjust centroid toward the anomaly boundary
        alpha = self.config.REFINEMENT_ALPHA  # default 0.05 — small step
        node.pattern_embedding = [
            node.pattern_embedding[i] * (1 - alpha) + content_embedding[i] *
alpha
            for i in range(len(node.pattern_embedding))
        ]
        node.return_to_infrastructure()
```

2.4 StructuralNodeLibrary

python

```
# nbsi/lifecycle/structural_library.py (continued)
from sklearn.metrics.pairwise import cosine_similarity

class StructuralNodeLibrary:
    def __init__(self):
        self.nodes: dict[str, StructuralNode] = {}

    def add(self, node: StructuralNode):
        self.nodes[node.node_id] = node

    def match(self, embedding: list, threshold: float = 0.75) -> list[tuple]:
        """
        Find all structural nodes that fire for the given embedding.
        Returns list of (node_id, similarity_score) sorted by score desc.
        Infrastructure nodes fire silently. Active nodes are being refined.
        """
        results = []
        for node in self.nodes.values():
            if node.pattern_embedding is None:
                continue
            sim = float(cosine_similarity([embedding], [node.pattern_embedding])
[0][0])
            if sim >= threshold:
                results.append((node.node_id, sim))
        return sorted(results, key=lambda x: x[1], reverse=True)

    def get_firing_nodes(self, embedding: list) -> list[StructuralNode]:
        """Return StructuralNode objects that fire for this embedding."""
        matches = self.match(embedding)
        return [self.nodes[nid] for nid, _ in matches]

    def infrastructure_cost(self) -> dict:
        """
        Metrics on library efficiency.
        Active nodes = those in reactivation (high cost).
        Infrastructure nodes = silent (near-zero cost).
        """
        total = len(self.nodes)
        active = sum(1 for n in self.nodes.values() if n.attention_mode ==
'active')
        return {'total': total, 'infrastructure': total - active, 'active':
```

```
active,
                'infrastructure_fraction': (total - active) / max(total, 1)}
```

2.5 Test Cases for the Lifecycle (Write These First)

> **TEST-DRIVEN REQUIREMENT**
> Write all four tests before implementing the lifecycle engine. They map directly to predictions 1-4 in the white paper.

python

```python
# tests/test_lifecycle.py

def test_observation_node_forms_on_novel_content():
    """Novel content creates observation node, not structural node."""
    engine = LifecycleEngine(StructuralNodeLibrary(), embedder, config)
    novel_embedding = embedder.encode(['quantum tunnelling probability
wavefunction'])[0]
    # Library is empty — no structural node should fire
    assert engine.library.match(novel_embedding.tolist()) == []
    obs_id = engine.observe('quantum tunnelling', novel_embedding.tolist())
    assert obs_id in engine._observation_nodes
    assert obs_id not in engine.library.nodes

def test_stabilisation_promotes_confirmed_diverse_node():
    """Node confirmed across structurally diverse sessions is promoted."""
    engine = LifecycleEngine(StructuralNodeLibrary(), embedder, config)
    obs_id = engine.observe('reliable recurrence', some_embedding)
    # Simulate MIN_CONFIRMATIONS confirmations
    for _ in range(config.MIN_CONFIRMATIONS):
        engine._observation_nodes[obs_id].update_pattern(some_embedding,
'recurrence')
    # Provide diverse degree distributions
    diverse_dists = [np.array([1,2,3,10]), np.array([1,1,5,5,8]),
np.array([2,2,2,2,9])]
    promoted = engine.evaluate_for_stabilisation(obs_id, diverse_dists)
    assert promoted == True
    assert obs_id in engine.library.nodes
    assert obs_id not in engine._observation_nodes

def test_anomaly_triggers_reactivation_not_error():
    """Content in anomaly zone reactivates infrastructure node."""
    engine = LifecycleEngine(StructuralNodeLibrary(), embedder, config)
    # Add a structural node directly
    node = StructuralNode(node_id='test-node', pattern_embedding=base_embedding)
    engine.library.add(node)
    assert node.attention_mode == 'infrastructure'
    # Eclipse content: similar but not quite matching
    eclipse_embedding = slightly_shifted(base_embedding)
    reactivated = engine.check_for_reactivation(eclipse_embedding)
    assert 'test-node' in reactivated
```

```
    assert engine.library.nodes['test-node'].attention_mode == 'active'

def test_refinement_returns_node_to_infrastructure():
    """After refine_reactivated, node returns to infrastructure mode."""
    engine = LifecycleEngine(StructuralNodeLibrary(), embedder, config)
    node = StructuralNode(node_id='test-node', pattern_embedding=base_embedding)
    node.reactivate()
    engine.library.add(node)
    engine.refine_reactivated('test-node', eclipse_embedding)
    assert engine.library.nodes['test-node'].attention_mode == 'infrastructure'
    assert engine.library.nodes['test-node'].reactivation_count == 1
```

3. Extended Session Lifecycle

The NBSISession is extended from v4.0 to integrate lifecycle engine calls at ingestion, query time, and session end. The two-layer separation is enforced here: the lifecycle engine operates on the structural layer; the session graph is the operational layer.

3.1 Full Session with Lifecycle Integration

```python
# nbsi/session/session.py
from nbsi.lifecycle.lifecycle_engine import LifecycleEngine
from nbsi.lifecycle.structural_library import StructuralNodeLibrary
# ... (other imports as per v4.0 spec)

class NBSISession:
    def __init__(self, session_id, config, embedder, llm_clients,
                 structural_library: StructuralNodeLibrary):
        self.session_id = session_id
        self.config = config
        self.embedder = embedder
        self.llm_clients = llm_clients
        self.graph_primary = ConceptGraph()
        self.graph_ensemble = None
        self.ensemble_mode = False
        self.stability_score = None
        self.active_speculations = {}
        # Lifecycle — shared structural library passed in, not owned by session
        self.lifecycle = LifecycleEngine(structural_library, embedder, config)
        self._session_degree_dist = None  # Captured at ingestion end

    async def ingest(self, text: str):
        chunks = chunk_text(text, self.config)
        all_extractions = []
        for chunk in chunks:
            raw = await extract_from_chunk(chunk, self.llm_clients)
            agreed = compute_agreement(raw, self.embedder)
            all_extractions.extend([{**r, 'edges': agreed} for r in raw[:1]])
        self.graph_primary = build_graph_from_extractions(all_extractions,
self.embedder)
        # Lifecycle: check each node against structural library
        self._apply_structural_shaping()
        self.graph_primary.prune_by_betweenness(self.config.MAX_NODES)
        self._session_degree_dist = self.graph_primary.degree_distribution()
        await self._check_stability(chunks)

    def _apply_structural_shaping(self):
        """
        For each session node, check against structural library.
        Structural nodes that fire shape the session graph — routing, weighting.
        Content does not enter the structural library here.
        Nodes that match nothing go to lifecycle observation.
        """
        for node in list(self.graph_primary._nodes.values()):
            if node.embedding is None:
```

```
                continue
            matches = self.lifecycle.library.match(node.embedding)
            if matches:
                # Structural node fires: boost activation of session node
                top_structural_sim = matches[0][1]
                node.activation = min(1.0, node.activation + top_structural_sim *
0.2)
                # Check for reactivation (anomaly zone)
                reactivated =
self.lifecycle.check_for_reactivation(node.embedding)
                for rid in reactivated:
                    self.lifecycle.refine_reactivated(rid, node.embedding)
            else:
                # No structural match: create or update observation node
                self.lifecycle.observe(node.label, node.embedding)

    def end_session(self):
        """
        Called after all queries are complete.
        1. Evaluate observation nodes for stabilisation.
        2. Reset operational layer completely.
        Structural layer (library) is NOT reset.
        """
        if self._session_degree_dist is not None:
            for obs_id in list(self.lifecycle._observation_nodes.keys()):
                obs = self.lifecycle._observation_nodes[obs_id]
                obs.sessions_active += 1
                # Collect historical distributions for this observation node
                # (stored in library's session log, not in the node itself)
                dists = self.lifecycle.library.get_session_distributions(obs_id)
                dists.append(self._session_degree_dist)
                self.lifecycle.evaluate_for_stabilisation(obs_id, dists)
        # Hard reset — operational layer destroyed
        self.reset()

    def reset(self):
        """Hard reset. All session state destroyed. Structural library
untouched."""
        self.graph_primary = ConceptGraph()
        self.graph_ensemble = None
        self.ensemble_mode = False
        self.stability_score = None
        self.active_speculations.clear()
        self._session_degree_dist = None
        # lifecycle._observation_nodes persists across sessions (they are
structural candidates)
        # lifecycle.library persists (it IS the structural layer)
```

3.2 The Generalisation Constraint Enforcement

CRITICAL ARCHITECTURAL CONSTRAINT

Structural nodes must not encode entity-specific patterns. The generalisation check must run before any observation node is promoted. A node that fires predominantly

for one entity's content must be flagged is_generalised=False and must not be promoted. This is the condition the obedience theorem depends on.

```python
# nbsi/lifecycle/lifecycle_engine.py (continued)

    def check_generalisation(self, obs_id: str,
                             entity_firing_counts: dict[str, int]) -> bool:
        """
        entity_firing_counts: {entity_id: count} across all sessions for this
node.
        If any single entity accounts for > ENTITY_SPECIFICITY_THRESHOLD of
firings,
        the node is entity-specific and must not be promoted.
        """
        total = sum(entity_firing_counts.values())
        if total == 0:
            return True
        max_entity_fraction = max(entity_firing_counts.values()) / total
        return max_entity_fraction < self.config.ENTITY_SPECIFICITY_THRESHOLD  #
default 0.6

    def _promote(self, obs: ObservationNode):
        # Generalisation check before promotion
        entity_counts = self.library.get_entity_counts(obs.node_id)
        if not self.check_generalisation(obs.node_id, entity_counts):
            # Entity-specific: do not promote. Keep in observation or prune.
            return
        structural = StructuralNode(
            node_id=obs.node_id,
            pattern_embedding=obs.pattern_embedding,
            session_count=obs.sessions_active,
            diversity_score=obs.diversity_score,
            is_generalised=True,
        )
        self.library.add(structural)
        del self._observation_nodes[obs.node_id]
```

4. Configuration Reference

Parameter	Default	Component	Description
CHUNK_SIZE	512	Ingestion	Tokens per chunk
CHUNK_OVERLAP	64	Ingestion	Overlap between chunks
MAX_NODES	200	Session Graph	Pruning target post-ingestion
MAX_PATH_DEPTH	5	Traversal	Max path length in beam search
BEAM_WIDTH	10	Traversal	Beam width
SEM_ALPHA	0.4	SEM	Max fractional weight change per speculation
SEM_DECAY_RATE	0.5	SEM	Exponential decay with graph distance
SEM_RADIUS	2	SEM	Max hop radius for SEM propagation
STABILITY_THRESHOLD	0.80	Ensemble	Ensemble mode activation threshold
OBSERVATION_MERGE_THRESHOLD	0.88	Lifecycle	Similarity to merge into existing observation node
MIN_CONFIRMATIONS	5	Lifecycle	Minimum confirmations before stabilisation evaluation
STABILISATION_THRESHOLD	0.40	Lifecycle	Composite score to promote to structural node
REACTIVATION_LOW_THRESHOLD	0.45	Lifecycle	Lower bound of anomaly zone (too low = no match)
REACTIVATION_HIGH_THRESHOLD	0.75	Lifecycle	Upper bound of anomaly zone (too high = clean match)
REFINEMENT_ALPHA	0.05	Lifecycle	Step size for centroid refinement on reactivation
ENTITY_SPECIFICITY_THRESHOLD	0.60	Lifecycle	Max fraction of firings from one entity before blocking promotion
DIVERSITY_THRESHOLD	0.30	Structural Library	Wasserstein diversity required for promotion

PARAMETER CALIBRATION NOTE

STABILISATION_THRESHOLD, REACTIVATION thresholds, and ENTITY_SPECIFICITY_THRESHOLD are free parameters that require empirical calibration across your target domains. The defaults are starting points. Phase 1 of the empirical programme (predictions 1-4 in the white paper) will produce measured values to replace them.

5. API Layer

5.1 Endpoints

Method	Path	Description
POST	/sessions	Create new session
DELETE	/sessions/{id}	Hard reset — destroys operational layer, structural library untouched
POST	/sessions/{id}/ingest	Ingest source text. Runs lifecycle shaping.
POST	/sessions/{id}/query	Query session graph
POST	/sessions/{id}/speculate	Add speculation — triggers SEM
POST	/sessions/{id}/confirm/{spec_id}	Confirm speculation
POST	/sessions/{id}/disconfirm/{spec_id}	Disconfirm + rollback
POST	/sessions/{id}/end	End session: evaluate stabilisation, then hard reset
GET	/sessions/{id}/state	Session metadata: stability, active specs, lifecycle state
GET	/library/nodes	List structural nodes with mode (infrastructure/active)
GET	/library/observations	List observation nodes with stabilisation scores
GET	/library/cost	Library efficiency metrics: infrastructure fraction, active count

5.2 Key Response Shapes

```json
# GET /library/cost
{
  'total_structural_nodes': 47,
  'infrastructure': 45,
  'active': 2,
  'infrastructure_fraction': 0.957,
  'observation_nodes': 12,
  'mean_stabilisation_score': 0.23
}

# POST /sessions/{id}/end
{
  'observation_nodes_evaluated': 12,
  'nodes_promoted': 1,
  'nodes_still_in_observation': 11,
  'session_graph_destroyed': true
}
```

```
# GET /sessions/{id}/state
{
  'session_id': '...',
  'stability_score': 0.83,
  'ensemble_mode': false,
  'active_speculations': 0,
  'session_nodes': 134,
  'structural_nodes_firing': 38,
  'observation_nodes_created_this_session': 4,
  'reactivations_this_session': 1
}
```

6. Build Order and Exit Criteria

Phase	Build	Exit Criterion
1	ConceptNode, ConceptEdge (base/effective split), ConceptGraph	Unit tests pass. apply_sem_delta changes only effective_weight not base_weight.
2	Chunker, Multi-LLM Extractor, Agreement Scorer	3-page document builds graph with nodes, edges, agreement scores populated.
3	Conductivity, Beam Search, Query Anchoring	Hand-built test graph returns top-k paths ranked by conductivity. Geometric mean matches manual calculation.
4	SEM: sem_propagate, sem_confirm, sem_rollback	All 4 SEM tests pass. Rollback tolerance < 1e-9.
5	Stability measurement, Ensemble mode	Stability in [0,1]. Ensemble uses labels not IDs. No-intersection warning fires correctly.
6	ObservationNode, StructuralNode, StructuralNodeLibrary	Unit tests pass. Observation node centroid updates without storing instances. Library match returns correct similarity ordering.
7	LifecycleEngine: observe, stabilise, reactivate, refine	All 4 lifecycle tests pass. Reactivation puts node in active mode. Refinement returns it to infrastructure.
8	Generalisation constraint enforcement	Entity-specific node (>60% from one entity) is not promoted. Generalised node is promoted correctly.
9	NBSISession with lifecycle integration	Integration test: ingest → lifecycle shaping fires → query → speculate → end_session → verify observation nodes evaluated → verify graph destroyed → verify library intact.
10	SessionManager, REST API	All endpoints return correct shapes. DELETE destroys graph, not library. GET /library/cost returns correct infrastructure fraction.

THE INTEGRATION TEST THAT PROVES THE TWO-LAYER SEPARATION WORKS

Ingest a document. Verify structural nodes fire and shape the session graph. Add a speculation and verify SEM changes effective_weight. End the session. Verify the session graph is fully destroyed (zero nodes, zero edges). Verify the structural library still contains all nodes with unchanged pattern embeddings. Verify no session content appears anywhere in the structural library. If this passes, the two-layer separation is correctly implemented.

7. Known Open Problems

Problem	Impact	Resolution path
STABILISATION_THRESHOLD not empirically derived — 0.40 is a starting estimate	Premature or delayed stabilisation affects library quality	Phase 1 empirical calibration: measure stabilisation scores for known-stable vs known-unstable patterns
Reactivation thresholds (0.45/0.75) define the anomaly zone — values not domain-validated	Zone too wide = excessive reactivation noise. Zone too narrow = missed refinements	Phase 1: inject known anomalies into test sessions, measure score distributions
Multiple simultaneous speculations: SEM deltas summed additively — interaction effects unspecified	Competing speculations may produce unpredictable effective_weight values	v1.1: specify multiplicative or max-pooling interaction
Wasserstein distance on degree distributions misses topology	Two graphs with identical degree distributions but different structure rated equally diverse	v2.0: replace with Weisfeiler-Lehman graph kernel
Observation node centroid convergence rate not validated	Centroid may not converge to stable representation before stabilisation threshold is crossed	Phase 1: convergence test — track centroid drift across confirmations
Correlated LLM hallucination not distinguished from genuine agreement	High agreement_score may reflect shared training bias not correctness	Ongoing — requires external ground truth corpus for calibration